Dudu Service Maker

Link submit: https://www.urionlinejudge.com.br/judge/en/problems/view/1610

Solution:

	Sử dụng visited và inPath	Sử dụng visited cải tiến
C++	http://ideone.com/97l4AA	http://ideone.com/NUHgdj
Java	http://ideone.com/pFdYj3	http://ideone.com/Mgt5ts
Python	http://ideone.com/Vhvlto	http://ideone.com/K6GfxK

Tóm tắt đề:

Cho một danh sách gồm N tài liệu cần chuẩn bị để hoàn thành nhiệm vụ. Trong đó một số tài liệu thì lại cần có tài liệu khác đi kèm. Hãy xác định xem danh sách này có chứa chu trình – tức tài liệu A cần có tài liệu B nào đó, đồng thời B lại phải cần có A – hay không.

Nói cách khác, cho một đồ thị có hướng N đỉnh, xác định xem có tồn tại chu trình trong đồ thị hay không.

Input:

Dòng đầu tiên chứa số nguyên *T* – số lượng test case của đồ thị ($T \le 100$).

Mỗi test case bắt đầu bằng hai số nguyên *N*, *M* ($1 \le N \le 10^4$, $1 \le M \le 3*10^4$) – số lượng tài liệu (số đỉnh) và số mối quan hệ (số cạnh của đồ thị).

M dòng tiếp theo, mỗi dòng chứa hai số nguyên *A*, *B* ($1 \le A, B \le N, A \ne B$) – tài liệu A phụ thuộc vào tài liệu B (hay có cạnh từ A đến B).

Output:

Với mỗi test case, nếu không tồn tại chu trình thì in ra "NAO", ngược lại in ra "SIM".

Ví dụ:

3 2 1 1 2 2 2 1 2 2 1 4 4	NAO SIM SIM

2 3 3 4 4 2 1 3	

Giải thích ví dụ:

Gồm 3 đồ thị:

Đồ thị 1 gồm 2 đỉnh và 1 cạnh (1, 2), không có cạnh từ 2 về 1 nên không tạo thành chu trình. Kết quả là NAO.

Đồ thị 2 gồm 2 đỉnh và 2 cạnh, tồn tại chu trình 1-2 nên kết quả là SIM.

Đồ thị 3 gồm 4 đỉnh và 4 cạnh, tồn tại chu trình 2-3-4 nên kết quả là SIM.

Hướng dẫn giải:

Nhận xét: Đồ thị chỉ có thể tồn tại chu trình chỉ khi nào có một cạnh nối từ u đến một đỉnh v nào đó được thăm trước đó, đồng thời từ v phải đến được u.

Do đó ta cần thêm một mảng để truy vết đường đi và kiểm tra điều kiện trên, tạm gọi là mảng path.

Như vậy, ý tưởng giải cơ bản sẽ là sử dụng DFS để duyệt qua từng đỉnh, với mỗi đỉnh u đang xét, ta duyệt qua từng đỉnh v kề với u:

- Nếu v chưa thăm thì ta duyệt DFS(v).
- Nếu v thăm rồi, lúc này cần kiểm tra trong mảng path xem từ v có đến được u hay không, nếu đến được thì chứng tỏ có chu trình.

Độ phức tạp: O(N) với mỗi đỉnh → **$O(N^2)$** cho toàn đồ thị.

Như vậy ta cần cải tiến để kiểm tra nhanh xem từ v có đến được u hay không.

Nhận xét: Nếu từ v đến được u thì v sẽ nằm trên đường đi từ gốc DFS đến u. Như vậy, nếu mình đánh dấu lại các đỉnh thuộc đường đi từ gốc đến u, thì có thể kiểm tra nhanh v có thuộc đường đi đó hay không, đồng nghĩa với việc từ v có đến được u hay không.

Do đó thay vì dùng mảng path, ta dùng một mảng là inPath với inPath[i] = true nếu i nằm trên đường đi từ gốc DFS đến đỉnh u đang xét, ngược lại inPath[i] = false.

Khi mình duyệt xong DFS(u), thì lúc trở về đỉnh cha của u, chắc chắn u không nằm trên đường đi từ gốc đến cha của u, nên cần gán lại inPath[u] = false trước khi thoát khỏi DFS(u).

Ngoài ra, còn một cách xử lý nữa là sử dụng mảng visited, nhưng thay vì lúc này chỉ đánh dấu 0/1 (false/true) thì lúc này mình đánh dấu 3 giá trị nhằm mục đích sử dụng nó để thực hiện chức năng của cả 2 mảng visited và path ở cách trên:

- visited[u] = 0 nếu u chưa được duyệt (tức visited[u] = false và inPath[u] = false theo cách vừa trình bày).
- visited[u] = 1 nếu u đã được duyệt và ta đang duyệt các đỉnh kề với u (visited[u] = true và inPath[u] = true).
- visited[u] = 2 nếu u đã được duyệt và đã duyệt xong các đỉnh kề với u (visited[u] = false và inPath[u] = false).

Độ phức tạp: **O(V + E)** với V, E lần lượt là số lượng đỉnh và số lượng cạnh của đồ thị. Tuy nhiên, cách sử dụng mảng visited 3 giá trị sẽ ít tốn bộ nhớ hơn.